LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-209462

Oracle Log Buffer Queueing

A. S. Rivenes

February 7, 2005

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

LAWRENCE LIVERMORE NATIONAL LABORATORY

Oracle Log Buffer Queueing

Andy Rivenes
Originally investigated, December 10, 2003

7000 East Avenue • L-089
Livermore, CA 94551

Table of Contents

Introduction

The purpose of this document is to investigate Oracle database log buffer queuing and its affect on the ability to load data using a specialized data loading system. Experiments were carried out on a Linux system using an Oracle 9.2 database. Previous experiments on a Sun 4800 running Solaris had shown that 100,000 entities per minute was an achievable rate. The question was then asked, can we do this on Linux, and where are the bottlenecks? A secondary question was also lurking, how can the loading be further scaled to handle even higher throughput requirements?

Testing was conducted using a Dell PowerEdge 6650 server with four CPUs and a Dell PowerVault 220s RAID array with 14 36GB drives and 128 MB of cache. Oracle Enterprise Edition 9.2.0.4 was used for the database and Red Hat Linux Advanced Server 2.1 was used for the operating system. This document will detail the maximum observed throughputs using the same test suite that was used for the Sun tests. A detailed description of the testing performed along with an analysis of bottlenecks encountered will be made. Issues related to Oracle and Linux will also be detailed and some recommendations based on the findings.

Testing Overview

The testing was conducted by splitting up the data provided by each source into nine relatively equal groups. The loading code was then run for each group simultaneously. This translated into nine Java loading processes running against the Oracle database. Once all nine jobs were running at full utilization (as evidenced by "top") three simultaneous 5 minute "snapshots" were taken. The first was an extended SQL trace of one of the nine jobs, the second was a beginning snapshot using a SQL query based on loading info within the application schema, and the third was a "Statspack" beginning snapshot. Once 5 minutes had elapsed from the start of the extended SQL trace, the trace was stopped and another application snapshot and a "Statspack" snapshot were taken. In several tests a second five minute interval was analyzed, using a different load job for the extended SQL trace.

Initially all nine jobs were traced, but once it was established that all nine jobs had similar resource and execution profiles this was cut back to just one as described above. As described, one or two 5 minute intervals were observed for each test. These were conducted after initial startup and once the jobs were running at a relative "steady state". On several tests, a "late" interval observation was conducted to see if the resource or execution profile was different after a significant amount of time had passed (e.g. to attempt to insure that no caching or other phenomenon affected the early part of the test) and it was found to be consistent throughout the test run(s).

Testing Sources

Three testing sources were used during each test run. The primary test source was the capturing of extended SQL trace data for one or more of the jobs running in the test. Typically a five minute interval was captured and the resulting trace file was then profiled with the Hotsos Profiler. From that output the "Interval Resource Profile" was used to evaluate the response time components and their time contribution. Figure 1 shows the interval resource profile for one of the tests performed (e.g. test 7a).

```
Interval Resource Profile
                                                  ------------ Duration Per Call ------------
Response Time Component                    Duration   # Calls         Avg         Min         Max
---------------------------- -------------------- --------- ----------- ----------- -----------
CPU service                   239.660000s   78.2%   220,524   0.001087s   0.000000s   3.490000s
log file sync                  38.596698s   12.6%    13,240   0.002915s   0.000002s   0.039492s
unaccounted-for                17.004281s    5.5%
SQL*Net message from client     4.956845s    1.6%       390   0.012710s   0.000091s   0.140028s
buffer busy waits               2.499641s    0.8%     2,504   0.000998s   0.000001s   0.078763s
latch free                      1.771481s    0.6%       490   0.003615s   0.000001s   0.038723s
SQL*Net more data from client   1.484256s    0.5%     6,933   0.000214s   0.000005s   0.114970s
enqueue                         0.487827s    0.2%       255   0.001913s   0.000001s   0.035624s
log file switch completion      0.023084s    0.0%         2   0.011542s   0.008676s   0.014408s
SQL*Net more data to client     0.006008s    0.0%        39   0.000154s   0.000017s   0.000742s
direct path read (lob)          0.004954s    0.0%       585   0.000008s   0.000000s   0.001991s
SQL*Net message to client       0.002308s    0.0%       390   0.000006s   0.000001s   0.000033s
db file sequential read         0.002193s    0.0%        27   0.000081s   0.000038s   0.000109s
direct path write               0.000904s    0.0%        78   0.000012s   0.000001s   0.000037s
buffer deadlock                 0.000054s    0.0%        22   0.000002s   0.000001s   0.000008s
---------------------------- -------------------- --------- ----------- ----------- -----------
Total                         306.500534s  100.0%
```

Figure 1. Interval resource profile, test 7a, file 15068.

The second test source was obtained from a snapshot procedure that is part of the application schema. This information approximates the number of "entities per minute" being loaded into the database for a given time interval. The third test source was obtained from Oracle's Statspack utility. This utility captures database statistic and wait event information for a given time interval. The information used from this utility was purely statistic based and included the number of system wide user commits and redo blocks written.

Hardware Configuration

As stated in the introduction, testing was conducted on a 4-way Dell PowerEdge 6650 server with a PowerVault 220s providing disk array storage. The PowerEdge server had 4 Intel Xeon 2.5 GHz CPUs with 1 MB cache, 16 GB of memory, 2 SCSI interfaces, and 1 NIC.

The PowerVault 220s had 14 36GB drives and 128MB of cache. The array was divided into 3 RAID 1 volumes for a total of 6 disks, and 1 RAID 1+0 4 disk wide volume for a total of 8 disks. Initially all logical drives were set with "write-thru" caching. The RAID 1 volumes were set with 64K stripe widths and the RAID 1+0 with 128K stripe width.

File System Layout

The following basic file system layout was used:

Mount point	Volume type	File system	Usage
/oracle	RAID 1	ext3	used for Oracle software (e.g. ORACLE_HOME), admin directories, local directories including application test data, and one control file (e.g. /oracle/oradata/SID/control02.ctl)

/ora01	RAID 1+0	ext3/raw	used for Oracle database files, other two control files. Raw and file system redo was tested, but not significantly different than RAID 1.
/ora02	RAID 1	ext3/raw	Used for redo log files. Was used for database files when the RAID 1+0 volume was used for redo log files.
/ora03	RAID 1	ext3/raw	Used for redo log files. Was used for database files when the RAID 1+0 volume was used for redo log files.

Linux Setup

Red Hat Advanced Server 2.1 Linux was installed with the enterprise kernel (e.g. Linux 2.4.9-e.3enterprise). It was initially thought that Oracle would be set up with large memory support (e.g. the target was a 12 GB SGA) and so the system was configured to allow this by using a shared memory file system (e.g. /dev/shm (shmfs)). No special kernel parameters were set other than those required by Oracle, specifically for semaphores and shared memory.

This test wasn't meant to be a comprehensive test on file system performance. However, because the resource profile of the test jobs had a large "log file sync"[1] component, the redo log setup was given particular scrutiny. Based on industry information/recommendations two flavors of file systems were tested for online redo log files, raw devices and ext3 file systems. For all other database files just the ext3 file system was used.

It is worth noting that several sources were found that indicated that raw device performance was sub-optimal in Linux and that the ext3 file system performed best for Oracle databases. In addition to raw vs. ext3 file systems, asynchronous I/O was also tested. Since Linux does not support file system direct I/O (e.g. unbuffered file system I/O) that feature could not be tested.

Once the redo log file I/O was identified as a significant bottleneck we also wanted to experiment with the speed of a solid state disk device in order to see if the elimination of the log file sync bottleneck would indeed drive up CPU usage, or result in some other unforeseen bottleneck. Since we didn't have access to a solid state disk system, this was simulated using "RAM" disks. This would not be an acceptable production setup since RAM disks would not protect from an instance failure with the loss of the UNIX memory system, but they were sufficient to simulate the latency and throughput of a solid state disk.

Oracle Setup

Oracle was installed using version 9.2.0.1 and patch set 9.2.0.4. A basic database was created with the following options:

- Enterprise Edition

[1] The Oracle wait event "log file sync" occurs when a user session COMMITs (or rolls back), the sessions redo information needs to be flushed to the redo logfile. The user session will post the LGWR to write all redo required from the log buffer to the redo log file. When the LGWR has finished it will post the user session. The user session waits on this wait event while waiting for LGWR to post it back to confirm all redo changes are safely on disk. – Oracle Note: 34592.1

- Oracle JVM
- Oracle Spatial
- Oracle InterMedia
- Oracle Text
- Oracle XML DB

Initially the database was installed with two 1 GB redo log files on dedicated RAID 1 devices (e.g. /ora02 and /ora03), and all database files on a RAID 1+0 volume (e.g. /ora01). A total of three control files were used, one control file was located on the /oracle volume along with the ORACLE_HOME software and ORACLE_BASE/admin directories, and the other two were located on the /ora01 volume (e.g. RAID 1+0).

The Oracle kernel was linked with the "async_on" option and the "disk_asynch_io" init.ora parameter was set to true initially. The SGA was set to approximately 800 MB (e.g. 400MB for database block buffers, 400 MB for the shared pool, and 1 MB for the log buffer). The database was created with a 4K block size to match the file system block size, and no additional block sizes or buffer pools were used. The init.ora option "pga_aggregate_target" was used instead of any hash or sort area sizing. See the "Parameters" section for a list of the non-default init.ora parameters used.

Testing Performed

Overview

Several initial tests were run and in most cases the individual job resource profile consisted of 60% - 70% CPU service and 10% - 20% "log file sync" or commit time. The rest of the time was spent among other wait events of less than 5% of total response time. From these initial tests it was clear that the loads consumed a large amount of CPU and waited a significant amount of time for commits. The actual workload consisted of a large number of inserts with additional selects and updates motivated by the inserts. In speaking with the principal developer, he concurred with this workload analysis.

In a typical response time optimization engagement, my recommendation would be to reduce the amount of CPU time and commits being performed. This follows the Hotsos Method R[2] approach of observing Amdahl's law and reducing the largest time consumers in descending order. However, in speaking with the application developer, the code is complex and will be costly to modify. He is aware of the limitations and enhancements are being planned. In the meantime, our goal remained unchanged, but the code was not changeable. This drove most of the testing to find ways to reduce the commit time (e.g. the duration of the log file sync wait events) in order to maximize CPU service time and thereby load the maximum number of entities possible. As an additional exercise, I made an attempt to quantify the theoretical maximums that might be possible given additional CPUs vs. additional nodes based on the queuing characteristics of the application (e.g. arrival rate and relative service times).

[2] Optimizing Oracle Performance, Pg. 20

Tests Performed

Twelve recorded tests were performed in all. The following describes the goals and results for each test:

Test	Environment	Goals	Results
1	1 job test using file system based redo logs	Attempt a baseline single user test and establish the maximum throughput for a single job.	Questionable validity. The commits per CPU second are twice the number of system wide commits per second that was recorded by Statspack.
2	9 job test using file based redo logs.	Attempt an initial load test.	Showed that log file sync times were indeed a significant bottleneck to increased throughput.
3	1 job test using RAM based redo log files.	Show the fastest possible loading in the absence of any significant bottlenecks.	Clearly showed that the latency of log file syncs was inhibiting throughput.
4	9 job test using RAM based redo log files.	Attempt the highest throughput load possible.	Achieved 100K entities per second at 100% CPU utilitization. log file sync accounted for only 5% of individual job response time and the bottleneck moved to the CPUs.
5	1 job test using RAID 1, file system based redo logs with asynchronous I/O.	Show the highest disk based throughput.	Showed a more reasonable correlation between system commit rate and individual job commit rates. I felt this test might have unduly benefited from the array cache so a 2 and 4 job test were performed later.
6	9 job test using RAID 1, file system based redo logs with asynchronous I/O.	Show the highest loaded disk based throughput.	Achieved 92K entities per minute. This is only 8% less than the max. Since the commit rate was the bottleneck the CPUs were not driven to 100%.
7 a/b	9 job tests using RAID 1, file system based redo logs with async I/O disabled in Oracle.	Show any differences between synchronous and asynchronous I/O.	A slight improvement over async I/O.

7 c/d	9 job tests using RAID 1, file system based redo logs with async I/O disabled in Oracle and the array write back cache set to write thru.	Show the benefit of the array cache for log writes.	Throughput was severely hampered. Clearly some kind of caching provides a huge benefit.
8	9 job test using RAID 1, file system based redo logs with asynchronous I/O.	Verify Test 7 a/b results.	Some problems were encountered performing the test, but the results were still impressive. The surprise was that async I/O hurt throughput.
9	9 job test using RAID 1, raw based redo log files with async I/O.	Show if raw based redo logs performed better.	Surprisingly raw redo log files performed worse. Normally we would expect them to be more efficient, but apparently Linux's raw I/O implementation is sub-optimal.
10	9 job test using RAID 1, raw based redo log files with synchronous I/O.	Same as Test 9, but with synchronous I/O.	There didn't appear to be a big difference, but synchronous I/O did appear to be worse. Since Linux does support kernelized asynchronous I/O this might be noteworthy if raw I/O was as fast as file system I/O.
11	2 job test using RAID 1, file system based redo logs with async I/O disabled in Oracle.	A 2 job test to try to show throughput with a minimum of queuing or load interference.	Numbers were slightly slower than the 1 user test, but appeared to validate baseline numbers.
12	4 job test using RAID 1, file system based redo logs with async I/O disabled in Oracle.	A 4 job test to try to show throughput with a some queuing. The hope was to validate the queuing models with additional data points.	Began to show degradation. Another datapoint to validate the queuing influences.

Additional Test Information

Data files were placed on a RAID 1+0 volume (e.g. /ora01). Initially it was thought that there would be I/O bottlenecks to the database files. As it turned out the only I/O bottleneck was with the redo log files. Tests were performed with redo log files on the RAID 1+0 volume with both raw and ext3 file systems. Results were not significantly different than RAID 1, and given the RAID 1+0 expense, it was decided not to pursue additional testing. In fact, RAID 1+0 is probably not necessary other than for the convenience of a large data file mount point. This application could probably make use of RAID 5 for the database files and save resources with no appreciable impact on performance.

Test Result Summary

Test Details					Trace file								Statspack				NE Stats
Test Number	Date	Start Time	Duration (sec)	Test Description	Trace File	Commits	CPU Service	Commits/ CPU sec	CPU Interval %	Number of log file sync	Avg duration	log file sync Interval %	Statspack Interval	commits/sec	Redo blks/sec	Redo blks / commit	entities/minute
1	11/24/03	12:32	323	1 job,	31973	18,030	175.82	102.55	54.5%	18,308	0.007459	42.3%	14 - 15	55	1,084.1	19.7	13,090.91
2a	11/24/03	12:43	340		1098	13,059	238.44	54.77	70.2%	12,869	0.005745	21.8%	16 - 17	355.9	6,697.3	18.8	84,449.57
2b	11/24/03	12:50	364		1091	12,566	236.22	53.20	64.9%	13,236	0.007934	28.8%	17 - 18	350.3	6,788.2	19.4	83,358.78
3	11/24/03	17:15	283	1 job test, redo on RAM disks	3291	27,593	272.80	101.15	96.5%	26,783	0.000038	0.4%	19 - 20	97.4	1,917.2	19.7	23,294.12
4a	11/24/03	17:36	272	RAM disk redo	7107	10,830	232.97	46.49	85.9%	9,493	0.001593	5.6%	21 - 22	422.4	8,299.2	19.7	100,751.68
4b	11/24/03	17:41	270	2nd 5 minutes	7097	10,136	228.40	44.38	84.8%	8,583	0.001464	4.7%	22 - 23	409.7	8,137.3	19.9	98,006.69
5	11/25/03	10:44	300	1 job test, RAID 1, fs	11265	25,117	246.18	102.03	82.3%	25,624	0.001091	9.3%	24 - 25	83.4	1,639.0	19.7	Missing
6a	11/25/03	10:55	287	RAID 1 redo, fs, async	12848	10,658	227.46	46.86	79.5%	10,030	0.003744	13.1%	26 - 27	389.7	7,557.9	19.4	92,178.34
6b	11/25/03	11:00	257	2nd 5 minutes	12854	9,360	206.12	45.41	80.4%	8,552	0.003918	13.1%	27 - 28	382.7	7,424.4	19.4	91,452.63
7a	11/25/03	13:28	307	RAID 1 redo, fs, no async	15068	11,924	239.66	49.75	78.2%	13,240	0.002915	12.6%	29 - 30	397.1	7,634.8	19.2	94,390.24
7b	11/25/03	13:33	705	2nd 5 minutes	15074	12,822	272.04	47.13	38.6%	13,864	0.003116	6.1%	30 - 31	394.1	7,586.1	19.3	93,928.93
7c	11/25/03	13:57	47	RAID 1 redo, fs, no async, write thru on array (no cache)	17192	933	20.92	44.60	45.2%	1,053	0.008425	19.2%	(only 1 interval for both tests)				(only 1 interval for both tests)
7d	11/25/03	13:58	296	Same test as 7c	17174	9,339	145.14	64.34	49.1%	9,024	0.012749	38.9%	32 - 33	297.5	5,509.8	18.5	70,464.00
8a	12/1/03	11:30	279	RAID 1 redo, fs, no async	6725	10,065	215.64	46.68	77.5%	9,765	0.002878	10.1%	34 - 35	383.1	7,551.7	19.7	91,722.77
8b	12/1/03	11:54	111	Note: This was taken at the end after some jobs had finished!	6727	5,998	93.79	63.95	85.0%	5,575	0.001493	7.5%	37 - 38	277.8	5,693.6	20.5	30,789.89
9a	12/1/03	17:14	240	RAID 1 redo, raw, async I/O	3511	8,648	168.10	51.45	70.2%	8,134	0.007014	23.8%	40 - 41	374.2	7,176.9	19.2	89,727.27
9b	12/1/03	17:18	362	2nd 5 minutes	3513	12,055	215.44	55.96	59.6%	13,685	0.007986	30.2%	41 - 42	342.1	6,645.7	19.4	81,391.30
10a	12/1/03	18:01	305	RAID 1 redo, raw, no async	6875	11,173	203.92	54.79	66.9%	11,518	0.007162	27.1%	43 - 44	368.9	6,955.4	18.9	87,662.34
10b	12/1/03	18:06	394		6883	14,076	256.31	54.92	64.9%	12,386	0.007419	23.3%	44 - 45	334.6	6,358.9	19.0	79,534.88
11a	12/2/03	16:48	24	2 job test, RAID 1 redo, fs, no async	3347	1,996	20.83	95.82	86.3%	2,340	0.001393	13.5%	46 - 47	176.8	3,412.60	19.3	41,907.12
11b	12/2/03	16:43	299	2 job test, RAID 1 redo, fs, no async	3349	24,075	262.64	91.67	87.9%	21,767	0.001286	9.4%	47 - 48	138.9	2,709.70	19.5	33,237.21
12a	12/2/03	17:01	271	4 job test, RAID 1 redo, fs, no async	4871	16,752	224.44	74.64	83.0%	18,550	0.001941	13.3%	49 - 50	276.6	5,299.40	19.2	65,981.42
12b	12/2/03	16:56	296	4 job test, RAID 1 redo, fs, no async	4874	17,041	239.45	71.17	81.1%	17,463	0.002376	14.0%	50 - 51	265.9	5,100.10	19.2	63,476.29

Findings

Commit Throughput

Based on initial tests performed, the main testing emphasis was placed on reducing commit time and therefore increasing commit throughput. The individual job average duration times for log file sync varied with load, and I believe this was due primarily to queuing and the caching constraints in the RAID array. The best fully loaded log file sync times achieved with disk devices was 2.9 milliseconds for average duration. At the system measurement level this translated to 2.5 milliseconds. I believe this slight discrepancy can be explained if we take in to account a slight overlap in I/Os since there were 9 independent jobs running. While it is true that this test did not take advantage of asynchronous I/O (and was still faster), if we look at the asynchronous I/O tests we see an even greater disparity between individual job duration times and overall system throughput (e.g. 3.7 milliseconds vs. 2.6 milliseconds). My best guess is that for LGWR writes the asynchronous code path consumes more time than performing the writes synchronously.

By far the best throughput was obtained using ram disks for redo logging (e.g. tests 3 and 4). This allowed the CPU utilization of the machine to approach 100% and the individual job response time component to approach 86% for CPU service. See Figure 2 for the queuing reduction differences.

In terms of redo configuration, it appears that RAID 1 devices, with the 128 MB of RAID array cache provided by the PowerVault 220s, and mounted as ext3 file systems with synchronous I/O are capable of supporting a maximum of about 400 commits/sec. This translates into an average service time of about 2.5 milliseconds. This results in an effective load rate of approximately 94K entities/minute. At this rate the CPUs are not at full utilization and the system is achieving approximately a 94% utilization rate[3]. See test 7 for these details. In order to achieve the additional 6% utilization it will require redo logging to solid state disk[4]. This will reduce the average duration of a "log file sync" low enough to reduce the queueing effects at the necessary arrival rate. If the "log file sync" service rate is not reduced, then adding more CPUs will only make the problem worse, not better. Figure 3 illustrates this phenomenon. If additional CPUs are added, the arrival rate of new work will move the system to the far right of the performance curve knee. The ability to meet the response time goal moves from 60% satisfaction to only 28% satisfaction. The reason our response time goal is only 60% satisfaction is that I based the response time on a commit rate to achieve 100K entities per minute. Since we can only achieve this with solid state disk, even our "best" disk solution cannot satisfy our response time goal.

In order to increase the single instance commit throughput, the log file sync service time must be reduced below a 2.5 milliseconds average service rate.

[3] This figure was derived by using the maximum rate obtained with RAM disk redo logs as the maximum rate possible (e.g. 100% utilization).

[4] Clearly using RAM disks is not an acceptable production solution. Since RAM disks reside in volatile system memory they cannot preserve the online redo log files in the event of a memory or machine failure and therefore we cannot guarantee recovery from instance failure.

Queueing Theory Multiserver Model

M/M/m 3.1e (2003/03/11)

name	$value_a$	$value_b$	unit	description
Units of measure				
jobunit		commit		workload unit (singular)
timeunit		sec		time unit (singular)
queueunit		instance		queue unit (singular)
serverunit		log file sync		service channel unit (singular)
serviceunit		sec/commit		service unit
throughputunit		commit/sec		throughput unit
Service level agreements				
λ	397	422	commit/sec	average arrival rate into the system
r_{max}	0.0025	0.0025	sec/commit	maximum tolerated response time
Architecture				
q	1	1	instance	number of instances
m	1	1	log file sync/inst	number of log file syncs per instance
μ	769	1050	commit/sec	average service rate
Performance forecasts				
color code	◆	—		graph color and shape code
model	1 x M/M/1	1 x M/M/1		Kendall notation
ρ	51.6%	40.2%		average utilization per log file sync
S	0.001300	0.000952	sec/commit	average service time
W	0.001387	0.000640	sec/commit	average queueing delay at specified λ
R	0.002687	0.001592	sec/commit	average response time at specified λ
CDF(r_{max})	**60.567%**	**79.195%**	satisfactions	% of jobs with $R \le r_{max}$ at specified λ
1 - *CDF*(r_{max})	**39.433%**	**20.805%**	dissatisfactions	% of jobs with $R > r_{max}$ at specified λ
Graph parameters				
λ_0		0	commit/sec	arrival rate axis minimum value
λ_1/max(λ_a, λ_b)		1.2		ratio of λ_1 to max(λ_a, λ_b)
λ_1		506.4	commit/sec	arrival rate axis maximum value
λ_Δ		25.32	commit/sec	average arrival rate increment

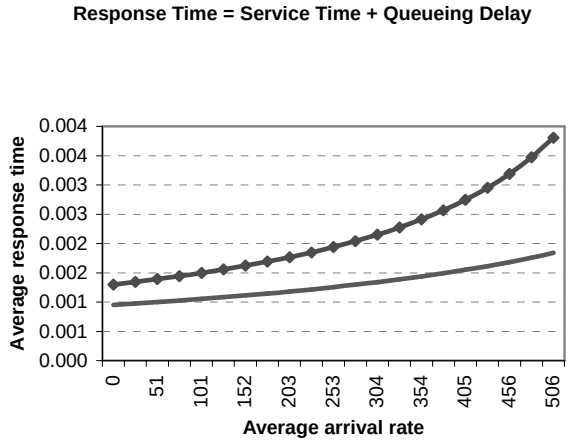

Figure 2. Redo logs on RAID 1, ext3 file system, synchronous I/O vs. RAM disks

Queueing Theory Multiserver Model

M/M/m 3.1e (2003/03/11)

name	$value_a$	$value_b$	unit	description
Units of measure				
jobunit		commit		workload unit (singular)
timeunit		sec		time unit (singular)
queueunit		instance		queue unit (singular)
serverunit		log file sync		service channel unit (singular)
serviceunit		sec/commit		service unit
throughputunit		commit/sec		throughput unit
Service level agreements				
λ	397	636	commit/sec	average arrival rate into the system
r_{max}	0.0025	0.0025	sec/commit	maximum tolerated response time
Architecture				
q	1	1	instance	number of instances
m	1	1	log file sync/inst	number of log file syncs per instance
μ	769	769	commit/sec	average service rate
Performance forecasts				
color code	◆	—		graph color and shape code
model	1 x M/M/1	1 x M/M/1		Kendall notation
ρ	51.6%	82.7%		average utilization per log file sync
S	0.001300	0.001300	sec/commit	average service time
W	0.001388	0.006218	sec/commit	average queueing delay at specified λ
R	0.002688	0.007519	sec/commit	average response time at specified λ
CDF(r_{max})	**60.545%**	**28.287%**	satisfactions	% of jobs with $R \le r_{max}$ at specified λ
1 - *CDF*(r_{max})	**39.455%**	**71.713%**	dissatisfactions	% of jobs with $R > r_{max}$ at specified λ
Graph parameters				
λ_0		0	commit/sec	arrival rate axis minimum value
$\lambda_1/\max(\lambda_a, \lambda_b)$		1.2		ratio of λ_1 to max(λ_a, λ_b)
λ_1		763.2	commit/sec	arrival rate axis maximum value
λ_Δ		38.16	commit/sec	average arrival rate increment

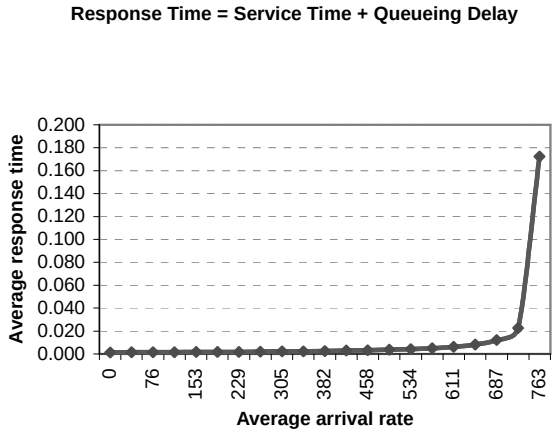

Figure 3. Commit throughput with increased workload.

CPU Capacity

The testing also allowed some modeling of the theoretical maximums that could be expected from both commit throughput and total CPU utilization based on the given configuration. From this modeling, and the corresponding testing results, it would appear that the maximum possible throughput is about 100,000 entities per minute for this four CPU machine. In achieving this rate the CPUs are at their maximum utilization and the queuing of requests passes the "knee"[5] in the performance curve. See Figure 4 and the results of Test 7a.

It would appear though, that with a low enough service time for commits (e.g. using a solid state disk) that a single system should continue to scale with the addition of more CPUs. Figure 5 illustrates the queuing affects of adding four additional physical processors (e.g. 8 logical processors for a total of 16 CPUs in the model) if we assume that the system can support the additional commit rate. The arrival rate of 423 was chosen since this is the number of commits per second that was observed at our peak transaction rate of 100K entities/second. We see that if we double the number of CPUs that we should be able to roughly double the number of commits per second to 846 and therefore double the number entities per second that we can process. This of course ignores any other bottlenecks that might show up as a result of the addition of more workload, but this option may be more cost effective than other alternatives, and could help offset the cost of a solid state disk device.

As a secondary note, the service rate of 69 commits per second was derived from the system wide commits per second statistic obtained from Statspack. This value, when divided by the number of jobs, provides an approximation of the commits per second per CPU occurring during the test(s). As the number of jobs goes up, this value begins to go down, presumably due to the effects of queueing. In the tests conducted this value didn't really change much between a 2 and 4 job test. In the 9 job tests it did. If we follow the response time curve in Figure 3 however, this is not an unexpected result. In the case of the value of 69, this is the value obtained from test 11b[6], which was the longer duration, and therefore probably more accurate, statistic collection for the 2 job test. Notice that this is approximately the same value that is obtained from test 12a, and close to 12b, which were taken from the 4 job test.

The question could be asked, how accurate is this? Can we really equate one of the load jobs to one CPU's utilization? I believe the answer is yes, and the reasoning is fairly simple. Oracle "background" processes (e.g. the server process for application connections) are single threaded. In the absence of I/O, network, or internal database contention, an Oracle process will consume service time on a single CPU. This is fairly simple to prove, and was observed during the testing. Single job tests consumed roughly 80% of one CPU. This was verified using "top", sar and vmstat. In addition, the run queue, as evidenced in sar and top was approximately 1. In other words, there was always one process running. Occasionally a background process or the process doing the monitoring would slip in, but from the operating system's perspective there was one process running consuming most of one CPU. This was true for the 2 and 4 job tests as well, with load and run queues corresponding to the number of jobs running. This is why I chose 9 jobs for the full load test. My expectation was

[5] Optimizing Oracle Performance, Pg. 257

[6] See Test Results for test details

that to maximize throughput there should be some CPU queuing, and therefore CPU saturation, if we could make the jobs efficient enough to be CPU bound. In fact, Oracle helped me confirm this CPU saturation once we switched to RAM disks for the redo log files. One of the reasons that extended SQL tracing was used as one of the test sources is that Oracle records all time used in a session. In fact, it will record wall clock, or elapsed time as well. In the case of CPU saturation however, there will be a difference between the elapsed time and the recorded time of the actions performed in the session. In fact, the elapsed time will be greater than the Oracle "response" time. This is due to the fact that Oracle does not perform timings unless it is running on a CPU or waiting on some event. In the case of a process waiting to run on the operating system CPU queue, there will be a discrepancy between the elapsed, or wall clock time, which will still be measuring time, and the timings Oracle has made for the sessions actions. This is called "unaccounted-for" time by Oracle, and others in the industry. In our case, the 9 job test with redo on RAM did in fact experience unaccounted-for time. It experienced at least 3.8% of unaccounted-for time during the test. See figure 6 for the details from the 7107 trace file.

Queueing Theory Multiserver Model

M/M/m 3.1e (2003/03/11)

name	$value_a$	$value_b$	unit	description
Units of measure				
jobunit		commits		workload unit (singular)
timeunit		sec		time unit (singular)
queueunit		system		queue unit (singular)
serverunit		CPU		service channel unit (singular)
serviceunit		sec/commits		service unit
throughputunit		commits/sec		throughput unit
Service level agreements				
λ	97.4	397.1	commits/sec	average arrival rate into the system
r_{max}	0.015	0.015	sec/commits	maximum tolerated response time
Architecture				
q	1	1	system	number of systems
m	8	8	CPU/system	number of CPUs per system
μ	69	69	commits/sec	average service rate
Performance forecasts				
color code	◆	—		graph color and shape code
model	1 x M/M/8	1 x M/M/8		Kendall notation
ρ	17.6%	71.9%		average utilization per CPU
S	0.014493	0.014493	sec/commits	average service time
W	0.000000	0.001952	sec/commits	average queueing delay at specified λ
R	0.014493	0.016445	sec/commits	average response time at specified λ
CDF(r_{max})	**64.477%**	**58.227%**	satisfactions	% of jobs with $R \le r_{max}$ at specified λ
1 - *CDF*(r_{max})	**35.523%**	**41.773%**	dissatisfactions	% of jobs with $R > r_{max}$ at specified λ
Graph parameters				
λ_0		0	commits/sec	arrival rate axis minimum value
$\lambda_1/\max(\lambda_a, \lambda_b)$		1.2		ratio of λ_1 to $\max(\lambda_a, \lambda_b)$
λ_1		476.52	commits/sec	arrival rate axis maximum value
λ_Δ		23.826	commits/sec	average arrival rate increment

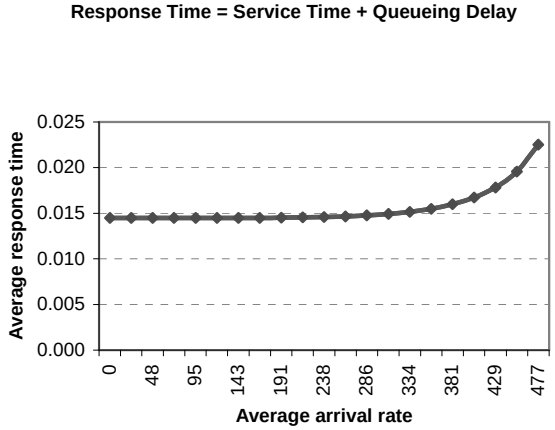

Figure 4 – CPU M/M/m queuing model

Queueing Theory Multiserver Model

M/M/m 3.1e (2003/03/11)

name	$value_a$	$value_b$	unit	description
Units of measure				
jobunit		commits		workload unit (singular)
timeunit		sec		time unit (singular)
queueunit		system		queue unit (singular)
serverunit		CPU		service channel unit (singular)
serviceunit		sec/commits		service unit
throughputunit		commits/sec		throughput unit
Service level agreements				
λ	423	846	commits/sec	average arrival rate into the system
r_{max}	0.015	0.015	sec/commits	maximum tolerated response time
Architecture				
q	1	1	system	number of systems
m	8	16	CPU/system	number of CPUs per system
μ	69	69	commits/sec	average service rate
Performance forecasts				
color code	◆	—		graph color and shape code
model	1 x M/M/8	1 x M/M/16		Kendall notation
ρ	76.6%	76.6%		average utilization per CPU
S	0.014493	0.014493	sec/commits	average service time
W	0.003010	0.000908	sec/commits	average queueing delay at specified λ
R	0.017502	0.015401	sec/commits	average response time at specified λ
CDF(r_{max})	**55.066%**	**61.617%**	satisfactions	% of jobs with $R \le r_{max}$ at specified λ
1 - *CDF*(r_{max})	**44.934%**	**38.383%**	dissatisfactions	% of jobs with $R > r_{max}$ at specified λ
Graph parameters				
λ_0		0	commits/sec	arrival rate axis minimum value
λ_1/max(λ_a, λ_b)		1.2		ratio of λ_1 to max(λ_a, λ_b)
λ_1		1015.2	commits/sec	arrival rate axis maximum value
λ_Δ		50.76	commits/sec	average arrival rate increment

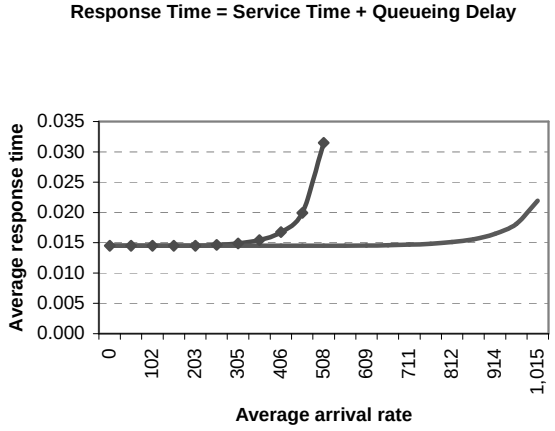

Figure 5. The effects of adding 4 more physical CPUs (total of 16 logical processors)

```
Interval Resource Profile
                                                 ------------ Duration Per Call --------------
Response Time Component                    Duration   # Calls         Avg          Min          Max
---------------------------- -------------------- --------- ----------- ----------- -----------
CPU service                   232.970000s   85.9%   211,800   0.001100s   0.000000s   4.720000s
log file sync                  15.119163s    5.6%     9,493   0.001593s   0.000002s   0.058383s
unaccounted-for                10.224881s    3.8%
SQL*Net message from client     6.329329s    2.3%       360   0.017581s   0.000122s   0.294809s
buffer busy waits               3.891499s    1.4%     2,555   0.001523s   0.000001s   0.043668s
latch free                      1.405972s    0.5%       437   0.003217s   0.000001s   0.032599s
enqueue                         0.737749s    0.3%       442   0.001669s   0.000001s   0.021420s
SQL*Net more data from client   0.425094s    0.2%     6,403   0.000066s   0.000005s   0.024065s
SQL*Net message to client       0.019176s    0.0%       360   0.000053s   0.000001s   0.015493s
log file switch completion      0.018860s    0.0%         2   0.009430s   0.006020s   0.012840s
SQL*Net more data to client     0.007279s    0.0%        36   0.000202s   0.000021s   0.000897s
direct path read (lob)          0.003030s    0.0%       543   0.000006s   0.000000s   0.000337s
db file sequential read         0.001162s    0.0%        14   0.000083s   0.000045s   0.000109s
direct path write               0.001137s    0.0%        72   0.000016s   0.000000s   0.000160s
buffer deadlock                 0.000109s    0.0%        37   0.000003s   0.000000s   0.000008s
---------------------------- -------------------- --------- ----------- ----------- -----------
Total                         271.154440s  100.0%
```

Figure 6. Resource Profile showing unaccounted-for time.

Entity Throughput Scalability

As a final wrap up on scalability, two more questions come to mind. How was the entity per minute rate affected as jobs were added, and how was the commit scalability affected? This might affect our decision on how many CPUs to place in each server. Based on the test result data from the tests with redo logs on RAID 1 volumes with ext3 file systems, synchronous I/O and write back cache enabled on the array, I plotted the entities per minute and the commits per CPU per job for 2, 4 and 9 job tests (e.g. the blue bars). See figures 7 and 8. The results show that it takes only 4 jobs to load 66K entities/minute, but over twice that to get to 94K. The scalability is linear between 2 and 4 jobs, but really falls off at 9. Unfortunately I should have run a 6 job test as well as an 8 job test to better pinpoint queuing affects at higher load levels (e.g. the knee of the performance curve). By adding the 9 job redo on RAM disk data we still see a large drop off (e.g. the "9–RAM" bar). At 9 jobs we know we were running to the right of the performance curve for commit rates for disk based redo and CPU queuing for RAM based redo.

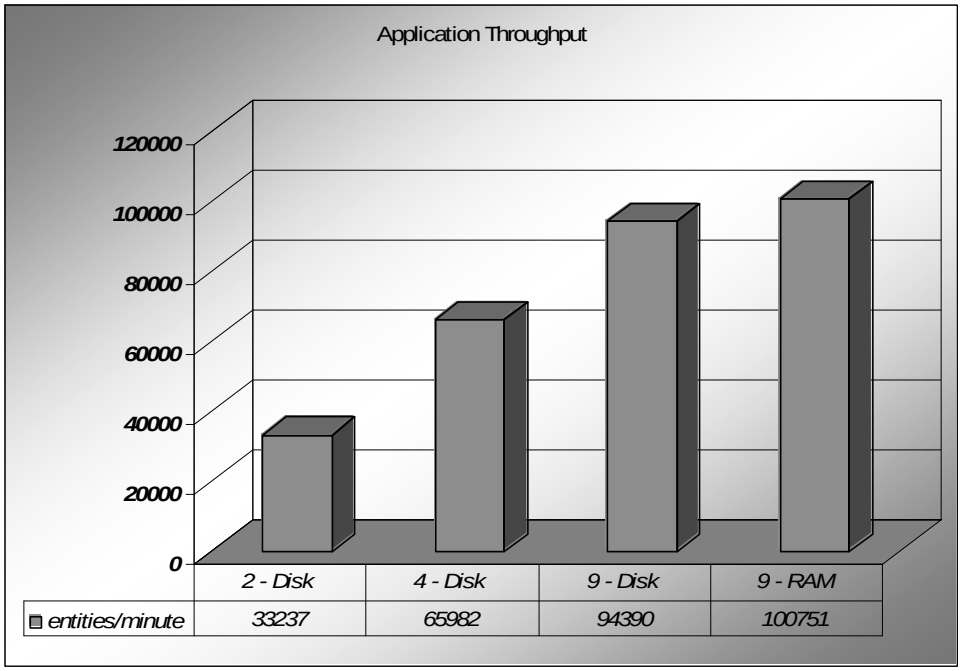

Figure 7. Entity throughput by number of jobs

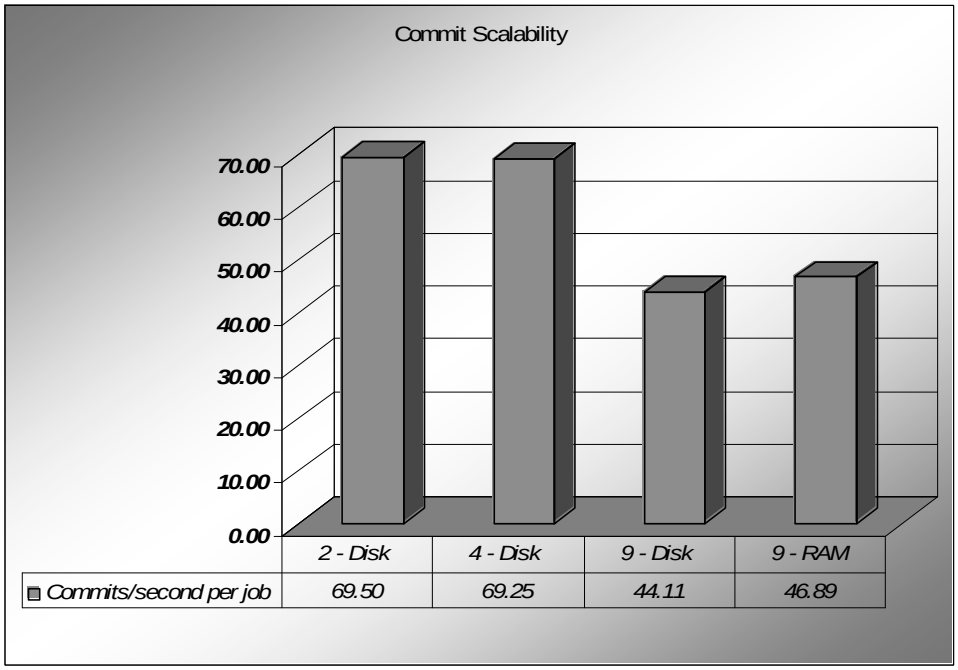

Figure 8. Commits per second per CPU

Additional Issues

Single Job Tests

A couple of anomalies occurred during testing that I don't have answers for. The first was the issue with one job tests. In all of the one job tests, commit rates were twice what two job tests were. Based on the resource profiles for the single vs. two job tests, I do not believe that this can be explained as contention within the database. Something else was happening that I haven't figured out. Interestingly enough, all the other tests seem to fall pretty much where you would expect in relation to each other (e.g. 2, 4, 9 jobs tests).

Raw Log File Throughput

In most UNIX systems, raw log file throughput will exceed that of file system throughput. Specifically, buffered file system throughput. This is due to the fact that Oracle writes redo log files in operating size blocks rather than file system sized blocks. In Linux this means that Oracle writes 512 byte blocks instead of 4K blocks. Normally, Oracle does not write 8 512 byte blocks every time it writes to the redo log files. During the tests performed, the average write size was twenty 512 byte blocks which means that Oracle was writing 2.5 file system size blocks every time a commit occurred. Since the operating system can't write half a block, it has to read in the third block in this case, modify the data, and write it back out. With a raw file the operating system just writes 20 blocks. Normally the other advantage of raw file I/O is that the operating system can make use of kernelized asynchronous I/O. Interestingly enough, none of this applied in the tests conducted, and synchronous file system I/O to the redo log files was the hands down performance winner.

Recommendations

Solid State Disk

Based on the workload and resource profile of the load jobs, using a solid state disk device for the online redo log files increases the overall system throughput. During testing, the use of RAM disks (e.g. simulated SSD) reduced the "log file sync" bottleneck considerably and provided a 6% boost in entity per minute throughput. In the long term this will also "deheat" the storage array by removing approximately half the total I/O, and this is foreground I/O (e.g. I/O that processes are directly waiting for).

Physical Memory

The machine had 16 GB of memory. Oracle used approximately 1 GB of that memory. The rest was used by processes, the OS, and the large majority by the UNIX file system cache. For this type of workload this could be scaled back significantly. I would expect that 4 GB would be sufficient.

Scalability

As part of the initial scope of this project, the question of what it would take to scale to even higher throughput requirements was asked. I believe that there are a couple of answers or guidelines that

could be followed. The first is that each server should have as many CPUs, and therefore load jobs on an approximate one to one basis, as the commit service rate can support. In our examples, this would be approximately 8 CPUs using disk based log files that can support an approximate service time of 2.5 ms. Even better, if solid state disk is used then even more CPUs/jobs could be supported and in the tested configuration all of the CPU capacity could be maximized.

In order to scale above the 94K – 100K entity/minute rate, the next logical step would be to try Oracle's Real Application Cluster (RAC) technology. Since the bottleneck(s) revolve around the commit rate and CPU service rate, and since these are instance wide limitations within Oracle, it makes since that adding additional instances should add additional scalability. At some point other bottlenecks will probably occur, but certainly this should allow for the additional load rate scaling of a single database loader.

Parameter Settings

init.ora Settings

System parameters with non-default values:

```
processes                = 150
timed_statistics         = TRUE
shared_pool_size         = 218103808
large_pool_size          = 33554432
java_pool_size           = 83886080
nls_language             = AMERICAN
nls_territory            = AMERICA
nls_sort                 = BINARY
nls_date_format          = MM/DD/YYYY
nls_numeric_characters   = .,
nls_timestamp_format     = YYYY-MM-DD"T"hh24:mi:ss.ff
disk_asynch_io           = FALSE
tape_asynch_io           = TRUE
control_files            = /ora01/oradata/SID/control01.ctl,
                           /oracle/oradata/SID/control02.ctl,
                           /ora01/oradata/SID/control03.ctl
db_block_buffers         = 204800
db_block_size            = 4096
db_writer_processes      = 1
compatible               = 9.2.0.0.0
db_file_multiblock_read_count= 16
fast_start_mttr_target   = 300
recovery_parallelism     = 4
control_file_record_keep_time= 21
undo_management          = AUTO
undo_tablespace          = UNDOTBS1
undo_retention           = 10800
remote_login_passwordfile= EXCLUSIVE
db_domain                = dbdhs
instance_name            = DBAG
service_names            = DBAG.dbdhs
serial_reuse             = ALL
session_cached_cursors   = 200
job_queue_processes      = 10
parallel_max_servers     = 0
hash_join_enabled        = TRUE
background_dump_dest     = /oracle/admin/SID/bdump
user_dump_dest           = /oracle/admin/SID/udump
core_dump_dest           = /oracle/admin/SID/cdump
optimizer_features_enable= 9.2.0
db_name                  = DBAG
open_cursors             = 300
optimizer_mode           = CHOOSE
star_transformation_enabled= FALSE
optimizer_max_permutations= 2000
optimizer_index_cost_adj = 50
optimizer_index_caching  = 80
query_rewrite_enabled    = TRUE
query_rewrite_integrity  = TRUSTED
pga_aggregate_target     = 209715200
workarea_size_policy     = AUTO
aq_tm_processes          = 1
```

References

Note: 34592.1, WAITEVENT: "log file sync" Reference Note, 11-NOV-2002, Oracle Corporation

Hotsos Profiler, Hotsos Enterprises, Ltd., www.hotsos.com

Optimizing Oracle Performance, September 2003, Cary Millsap with Jeff Holt, O'Reilly

Oracle File System Integration and Performance, January 2001, Richard McDougall, Sriram Gummuluru, Sun Microsystems

Evaluation of VERITAS File System to Enhance Oracle RDBMS Performance, Version 1.0, August 23, 1999, Roby Sherman

File System Performance White Paper, Part 1, 1996, VERITAS Software Corporation

File System Performance White Paper, Part 2, 1996, VERITAS Software Corporation

File System Performance White Paper, Part 3, 1996, VERITAS Software Corporation

Tips and Techniques: Install and Configure Oracle9i on Red Hat Linux Advanced Server, An Oracle White Paper, August 2002, Oracle Corporation

Tuning Oracle Database Server and Linux, Part 1, Bert Scalzo, Quest Software

Tuning an Oracle8i Database Running Linux, Part 2, Bert Scalzo, Quest Software

LINUX Maximus Part 2: The RAW Facts on File Systems, Bert Scalzo, Quest Software